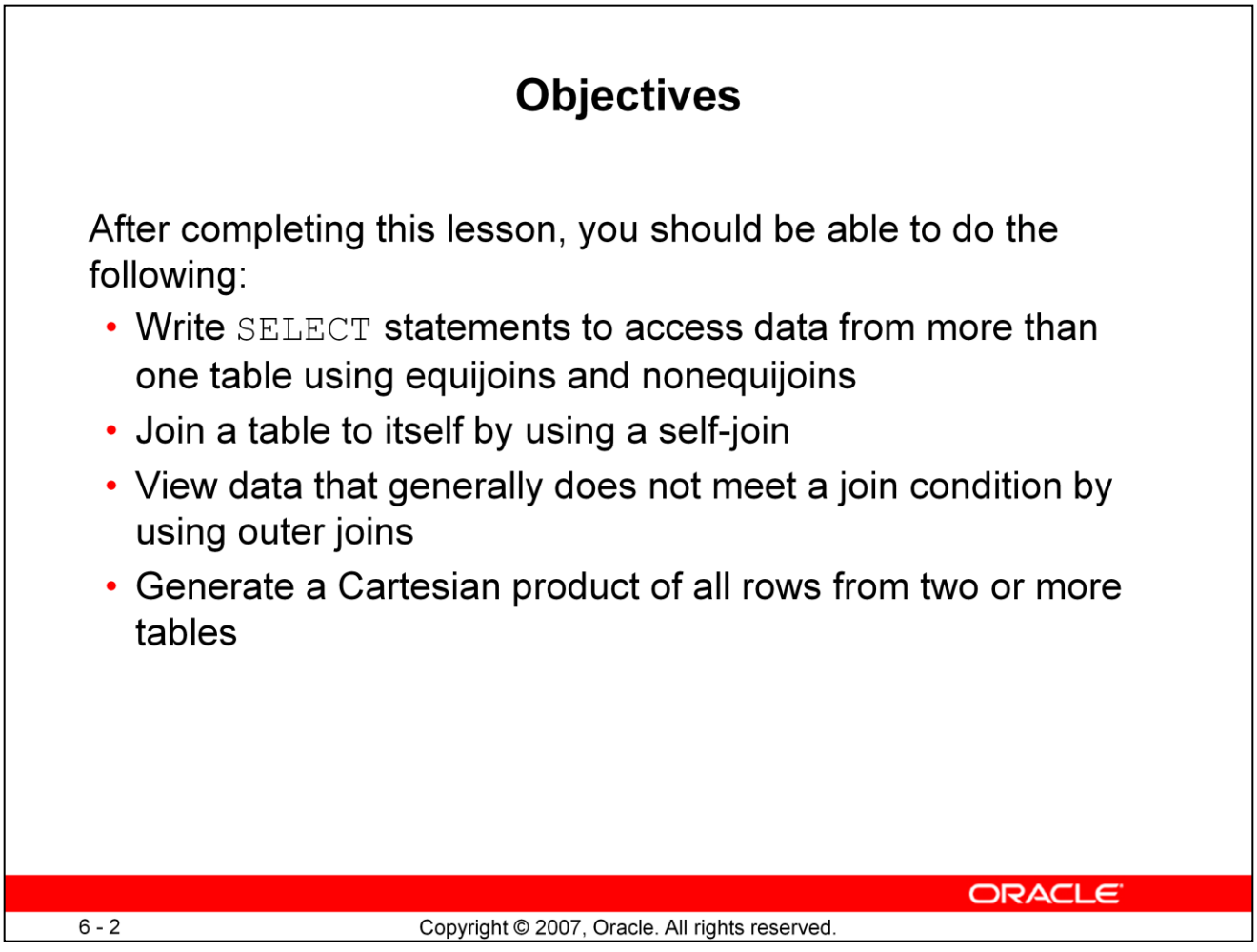
6

Displaying Data from Multiple Tables

ORACLE



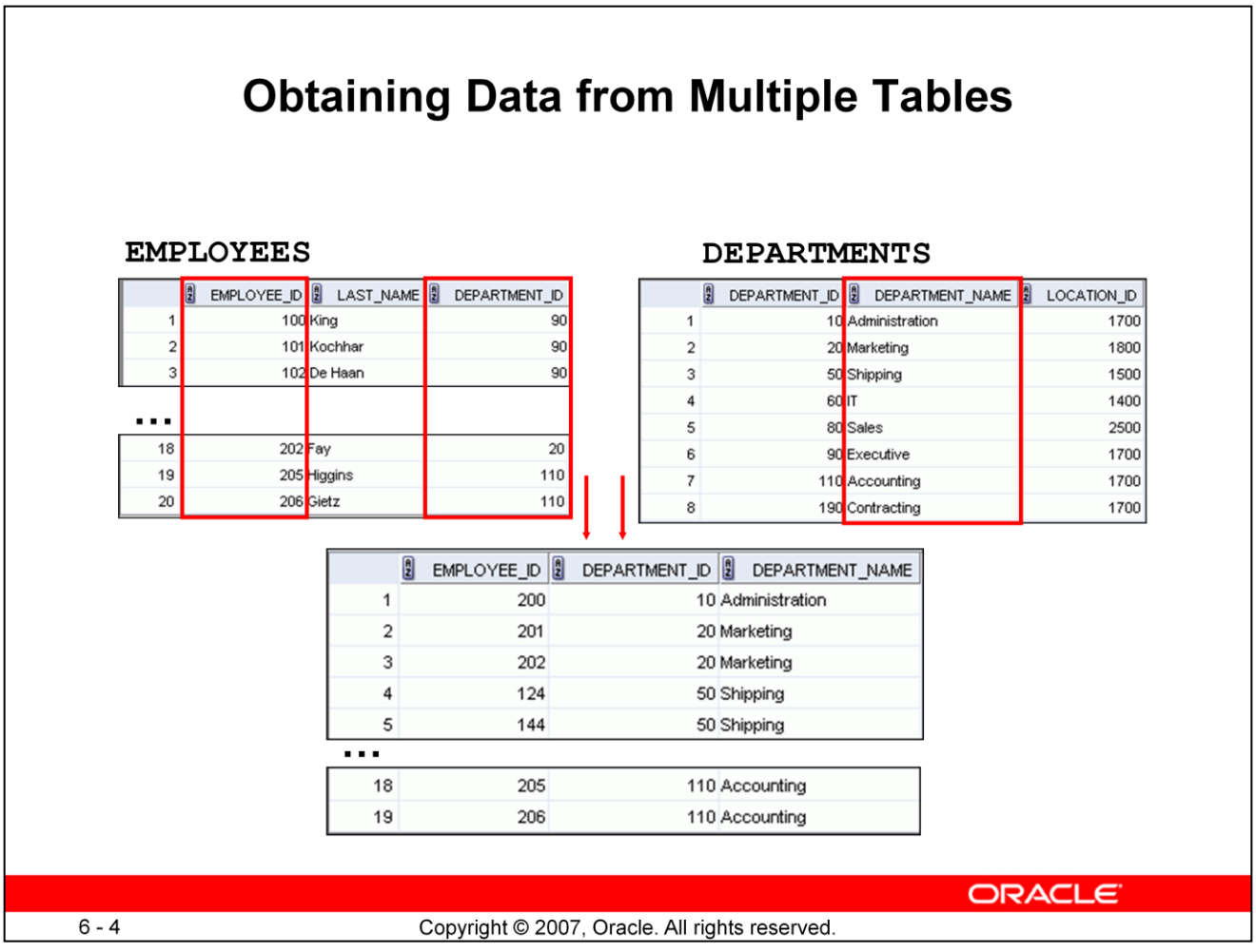
Objectives

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

Note: Information on joins is found in the section on *SQL Queries and Subqueries: Joins* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join

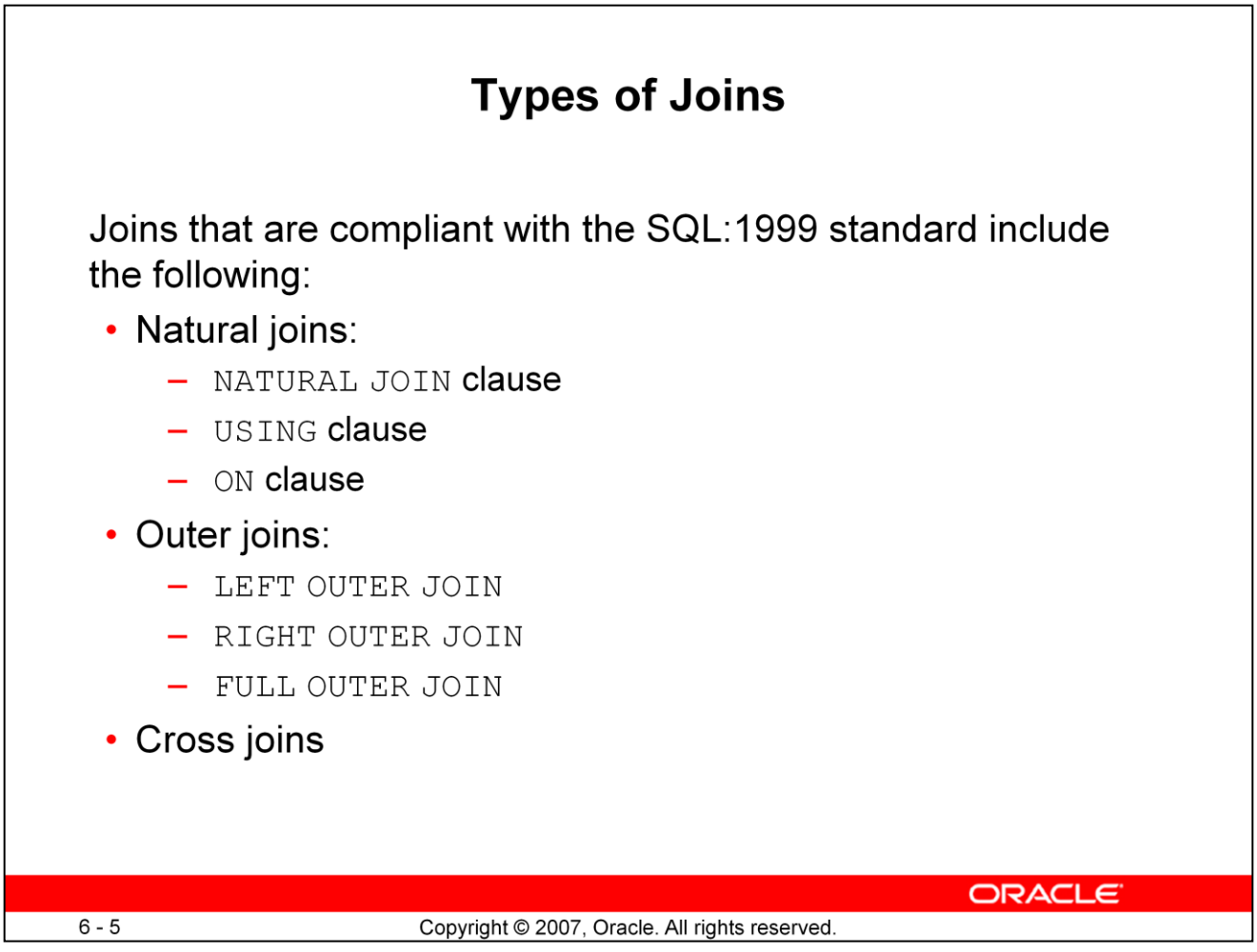


Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the example in the slide, the report displays data from two separate tables:

- Employee IDs exist in the `EMPLOYEES` table.
- Department IDs exist in both the `EMPLOYEES` and `DEPARTMENTS` tables.
- Department names exist in the `DEPARTMENTS` table.

To produce the report, you need to link the `EMPLOYEES` and `DEPARTMENTS` tables, and access data from both of them.

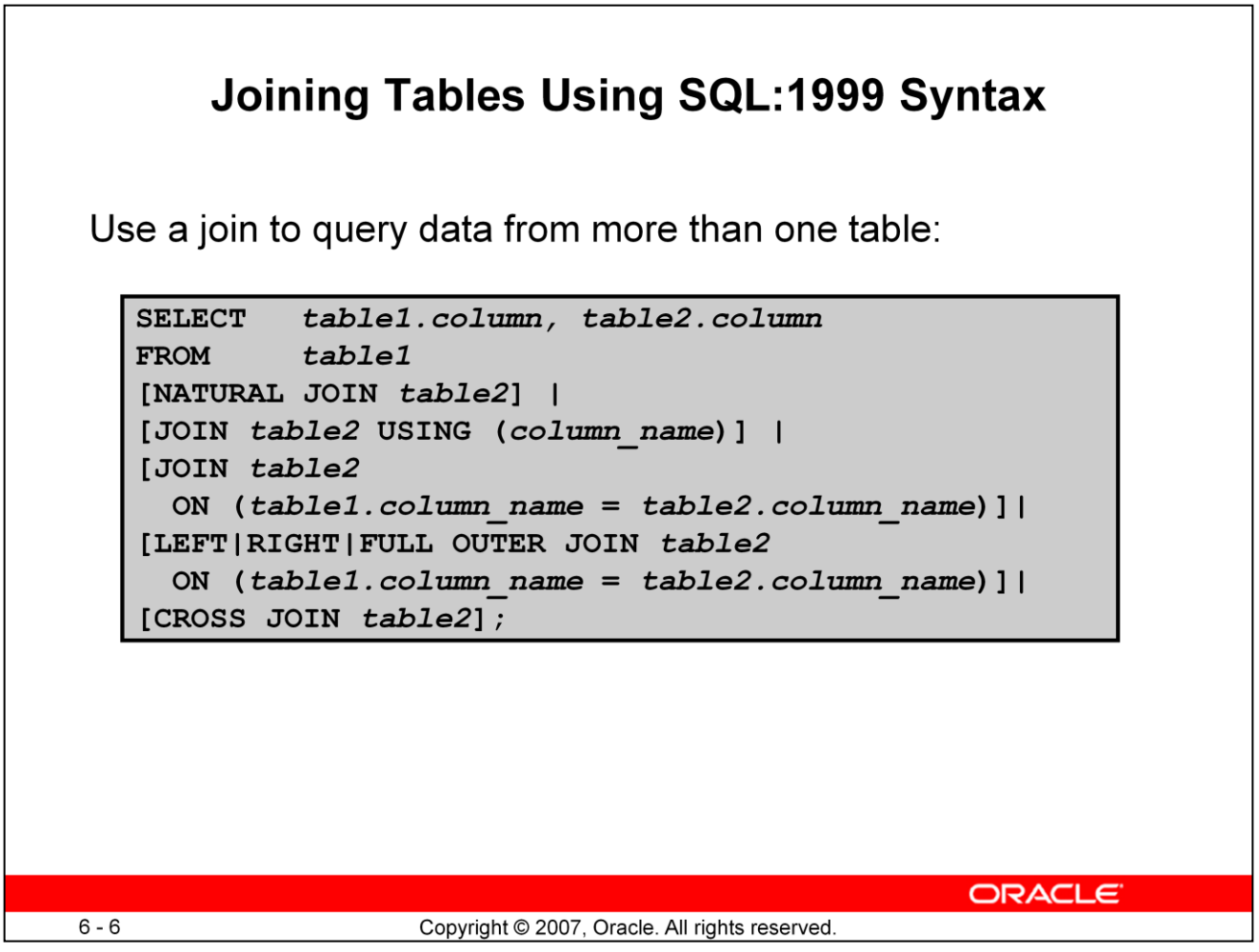


Types of Joins

To join tables, you can use a join syntax that is compliant with the SQL:1999 standard.

Note: Before the Oracle9*i* release, the join syntax was different from the American National Standards Institute (ANSI) standards. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in the prior releases. For detailed information about the proprietary join syntax, see Appendix C: Oracle Join Syntax.

Note: The following slide discusses the SQL:1999 join syntax.



Joining Tables Using SQL:1999 Syntax

In the syntax:

table1.column denotes the table and the column from which data is retrieved

NATURAL JOIN joins two tables based on the same column name

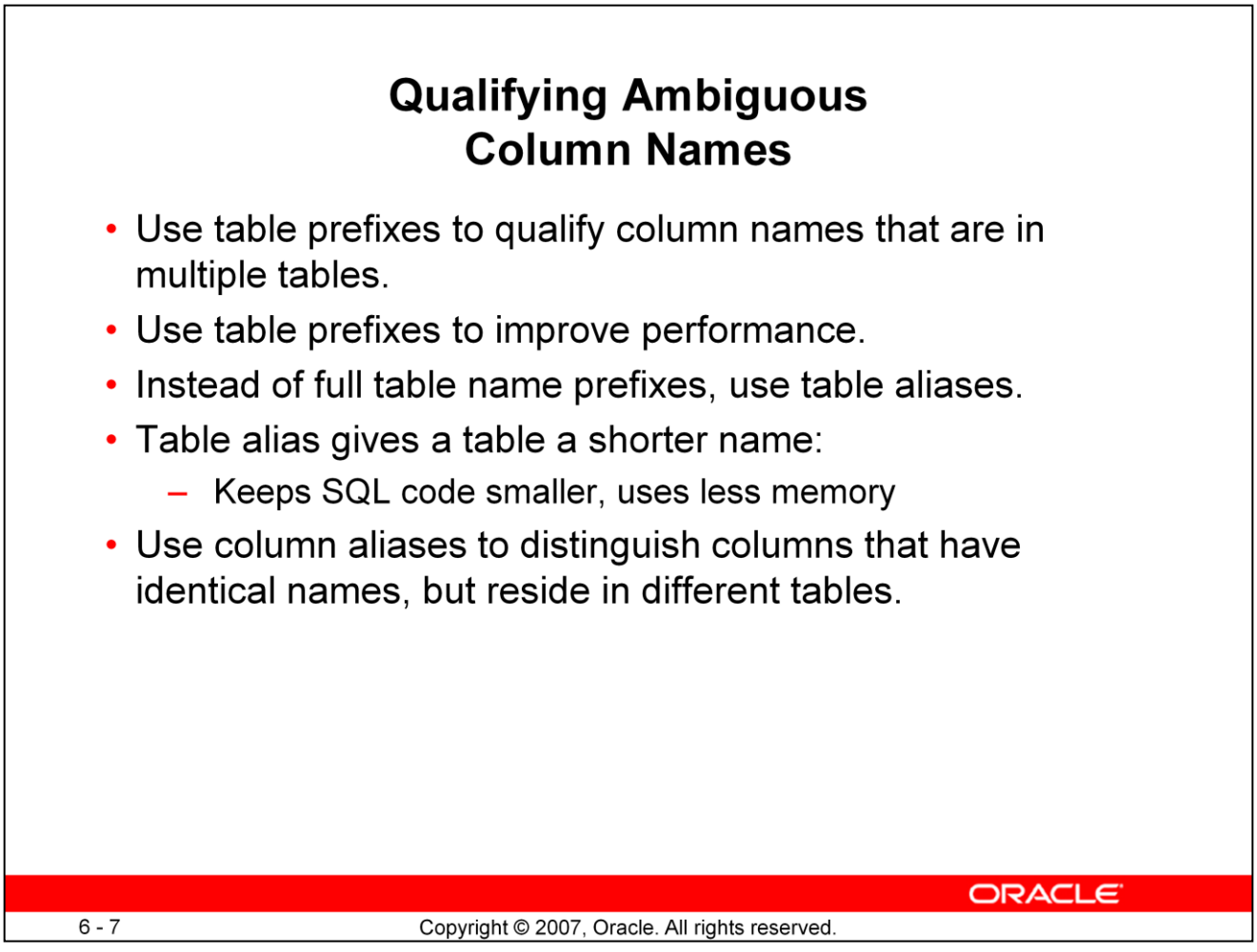
JOIN *table2* USING *column_name* performs an equijoin based on the column name

JOIN *table2* ON *table1.column_name* = *table2.column_name* performs an equijoin based on the condition in the ON clause

LEFT/RIGHT/FULL OUTER is used to perform outer joins

CROSS JOIN returns a Cartesian product from the two tables

For more information, see the section titled *SELECT* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*



Qualifying Ambiguous Column Names

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column in the `SELECT` list could be from either the `DEPARTMENTS` table or the `EMPLOYEES` table. It is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using the table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

However, qualifying column names with table names can be time consuming, particularly if the table names are lengthy. Instead, you can use *table aliases*. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore using less memory.

The table name is specified in full, followed by a space and then the table alias. For example, the `EMPLOYEES` table can be given an alias of `e`, and the `DEPARTMENTS` table an alias of `d`.

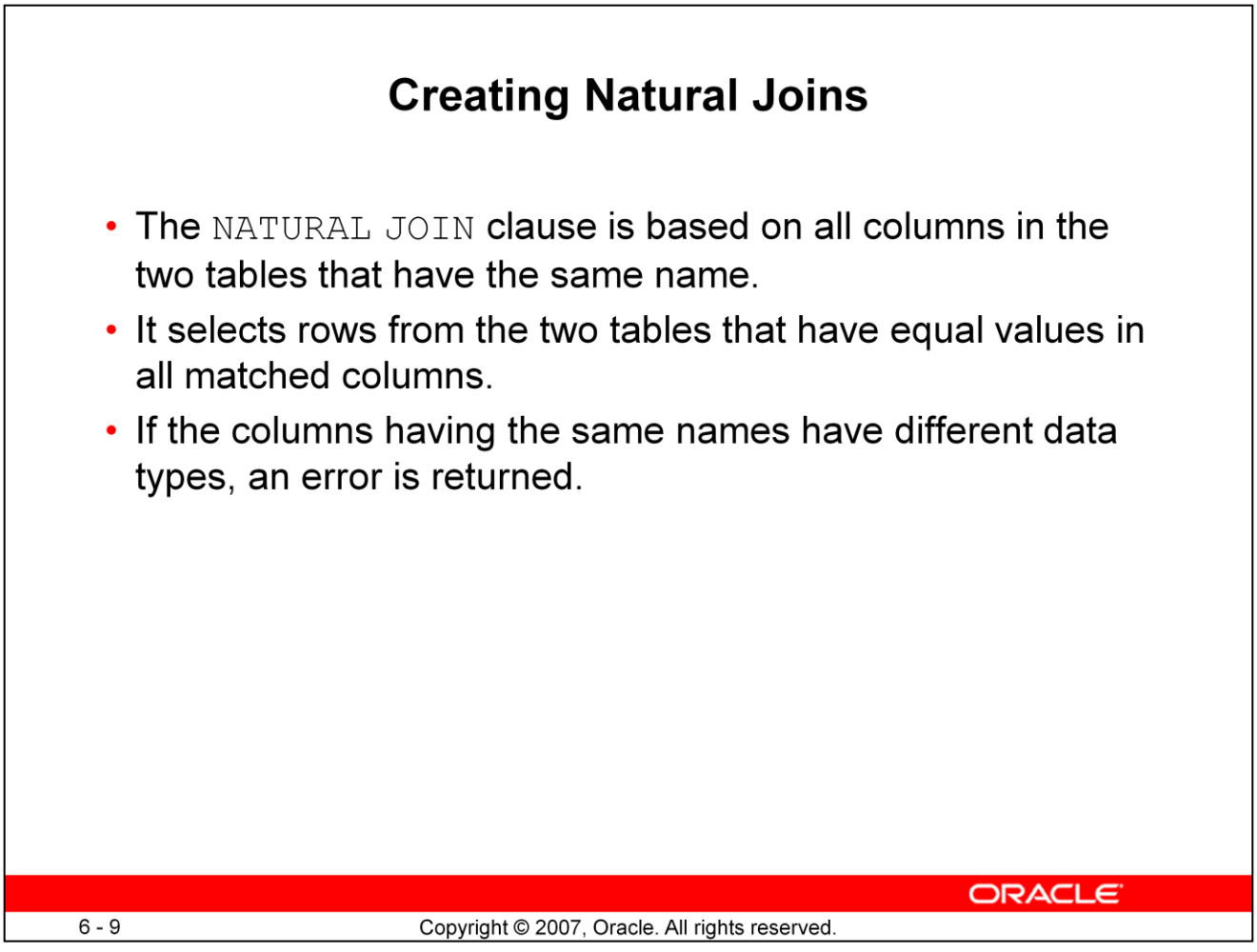
Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the `FROM` clause, then that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.

- The table alias is valid for only the current `SELECT` statement.

Lesson Agenda

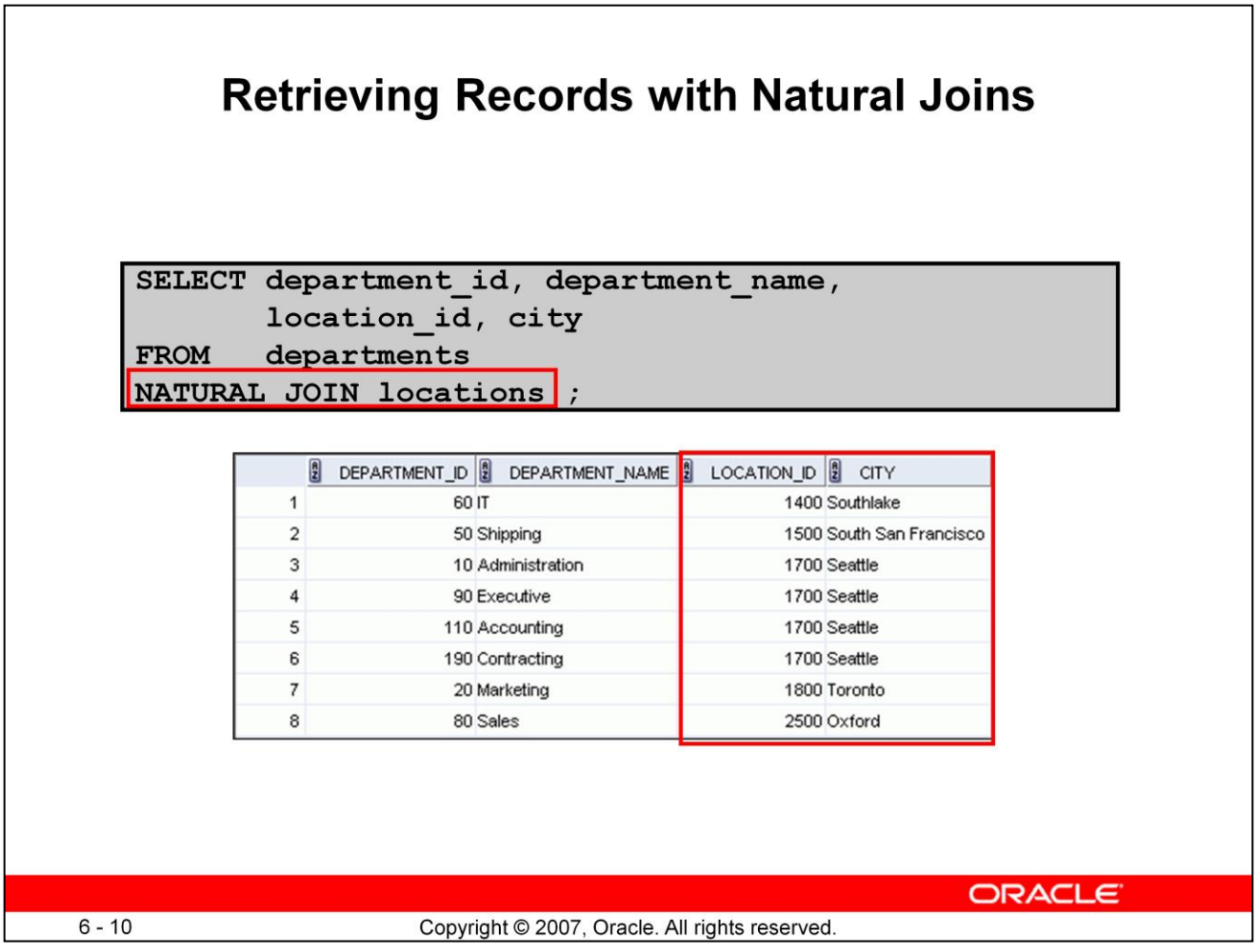
- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join



Creating Natural Joins

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the `NATURAL JOIN` keywords.

Note: The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, then the `NATURAL JOIN` syntax causes an error.



Retrieving Records with Natural Joins

In the example in the slide, the `LOCATIONS` table is joined to the `DEPARTMENT` table by the `LOCATION_ID` column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Natural Joins with a `WHERE` Clause

Additional restrictions on a natural join are implemented by using a `WHERE` clause. The following example limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT  department_id, department_name,
        location_id, city
FROM    departments
NATURAL JOIN locations
WHERE   department_id IN (20, 50);
```

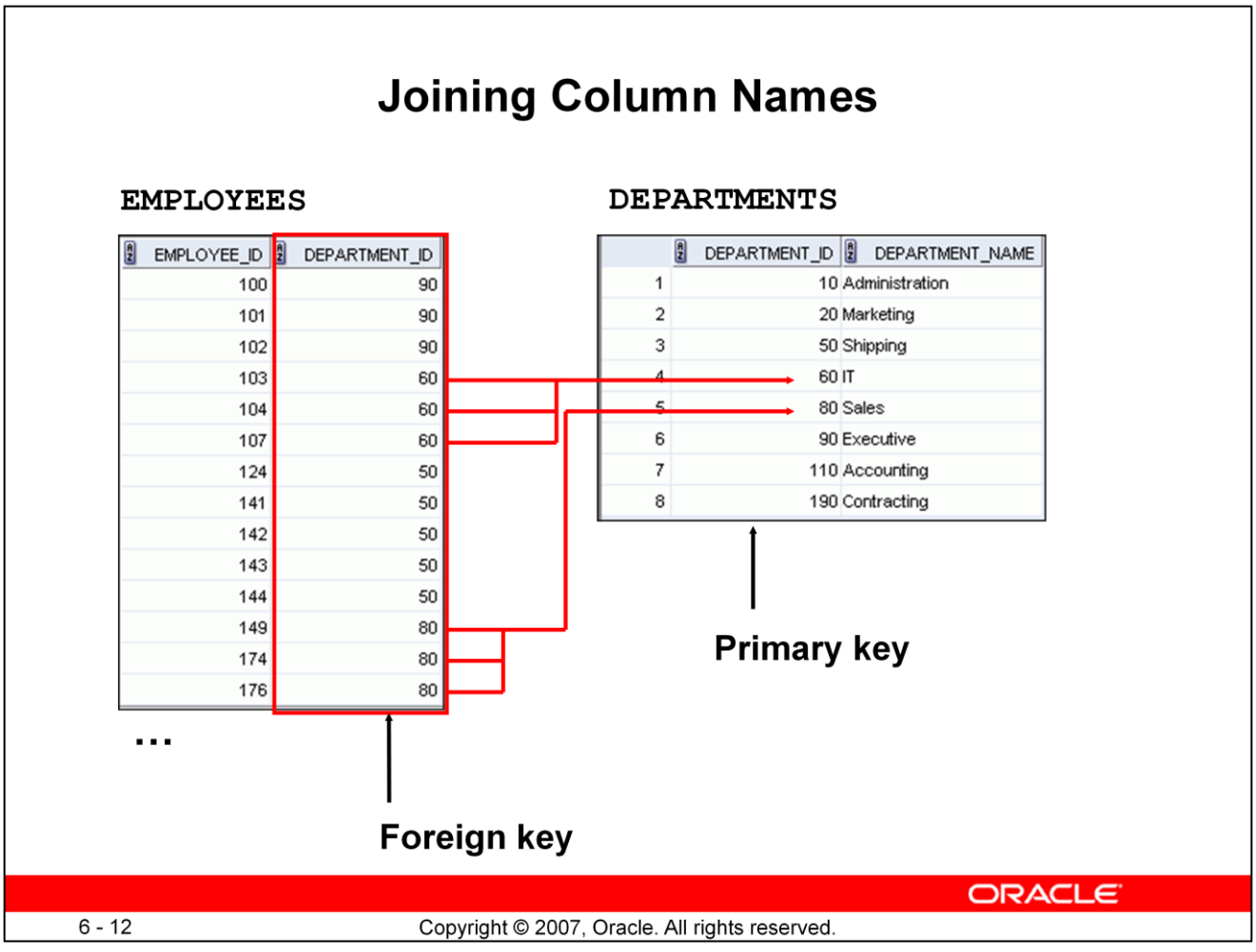
Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, natural join can be applied using the USING clause to specify the columns that should be used for an equijoin.
- Use the USING clause to match only one column when more than one column matches.
- The NATURAL JOIN and USING clauses are mutually exclusive.

ORACLE

Creating Joins with the USING Clause

Natural joins use all columns with matching names and data types to join the tables. The USING clause can be used to specify only those columns that should be used for an equijoin.



Joining Column Names

To determine an employee's department name, you compare the value in the `DEPARTMENT_ID` column in the `EMPLOYEES` table with the `DEPARTMENT_ID` values in the `DEPARTMENTS` table. The relationship between the `EMPLOYEES` and `DEPARTMENTS` tables is an *equijoin*; that is, values in the `DEPARTMENT_ID` column in both the tables must be equal. Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins* or *inner joins*.

Retrieving Records with the USING Clause

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

ORACLE

6 - 13

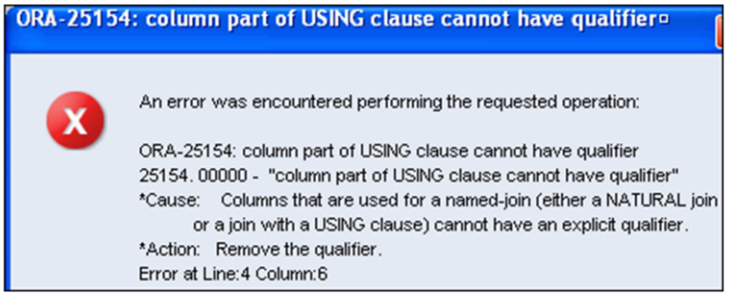
Retrieving Records with the USING Clause

In the example in the slide, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS tables are joined and thus the LOCATION_ID of the department where an employee works is shown.

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```



ORACLE

Using Table Aliases with the USING clause

When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it. For example, in the query mentioned in the slide, you should not alias the location_id column in the WHERE clause because the column is used in the USING clause.

The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement. For example, the following statement is valid:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

Because, other columns that are common in both the tables, but not used in the USING clause, must be prefixed with a table alias otherwise you get the “column ambiguously defined” error.

In the following statement, manager_id is present in both the employees and departments table and if manager_id is not prefixed with a table alias, it gives a “column ambiguously defined” error.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
```

```
WHERE  department_id = 50;
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

ORACLE

Creating Joins with the ON Clause

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400

...

ORACLE

Retrieving Records with the ON Clause

In this example, the `DEPARTMENT_ID` columns in the `EMPLOYEES` and `DEPARTMENTS` table are joined using the `ON` clause. Wherever a department ID in the `EMPLOYEES` table equals a department ID in the `DEPARTMENTS` table, the row is returned. The table alias is necessary to qualify the matching `column_names`.

You can also use the ON clause to join columns that have different names. The parenthesis around the joined columns as in the slide example, `(e.department_id = d.department_id)` is optional. So, even `ON e.department_id = d.department_id` will work.

Note: SQL Developer suffixes a '_1' to differentiate between the two `department_ids`.

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

ORACLE

Creating Three-Way Joins with the ON Clause

A three-way join is a join of three tables. In SQL:1999–compliant syntax, joins are performed from left to right. So, the first join to be performed is `EMPLOYEES JOIN DEPARTMENTS`. The first join condition can reference columns in `EMPLOYEES` and `DEPARTMENTS` but cannot reference columns in `LOCATIONS`. The second join condition can reference columns from all three tables.

Note: The code example in the slide can also be accomplished with the `USING` clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id)
```

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
WHERE   e.manager_id = 149 ;
```

ORACLE

Applying Additional Conditions to a Join

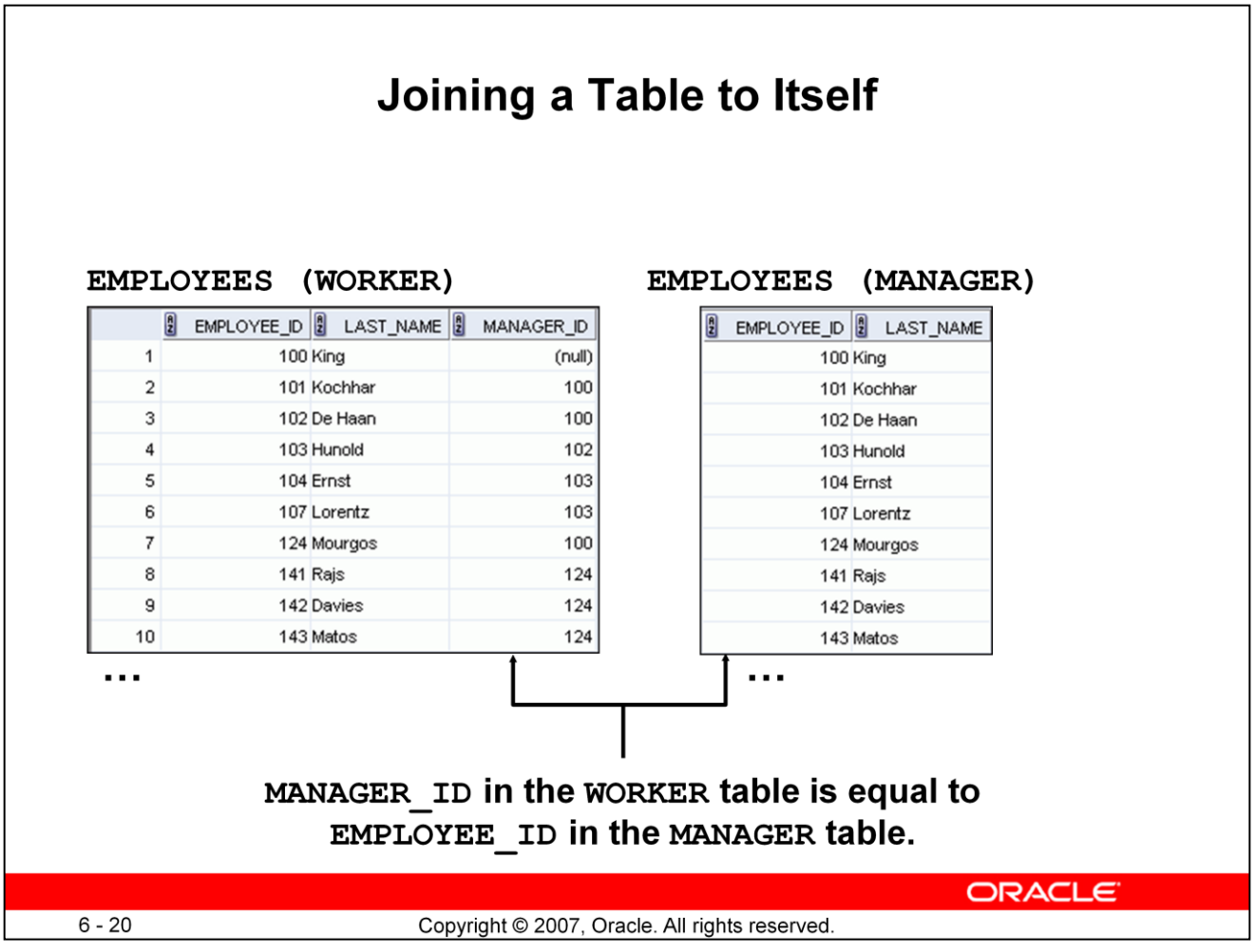
You can apply additional conditions to the join.

The example shown performs a join on the EMPLOYEES and DEPARTMENTS tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join



Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self-join. For example, to find the name of Lorentz's manager, you need to:

- Find Lorentz in the EMPLOYEES table by looking at the LAST_NAME column
- Find the manager number for Lorentz by looking at the MANAGER_ID column. Lorentz's manager number is 103.
- Find the name of the manager with EMPLOYEE_ID 103 by looking at the LAST_NAME column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the LAST_NAME column and the MANAGER_ID value of 103. The second time you look in the EMPLOYEE_ID column to find 103 and the LAST_NAME column to find Hunold.

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

ORACLE

6 - 21

Self-Joins Using the ON Clause

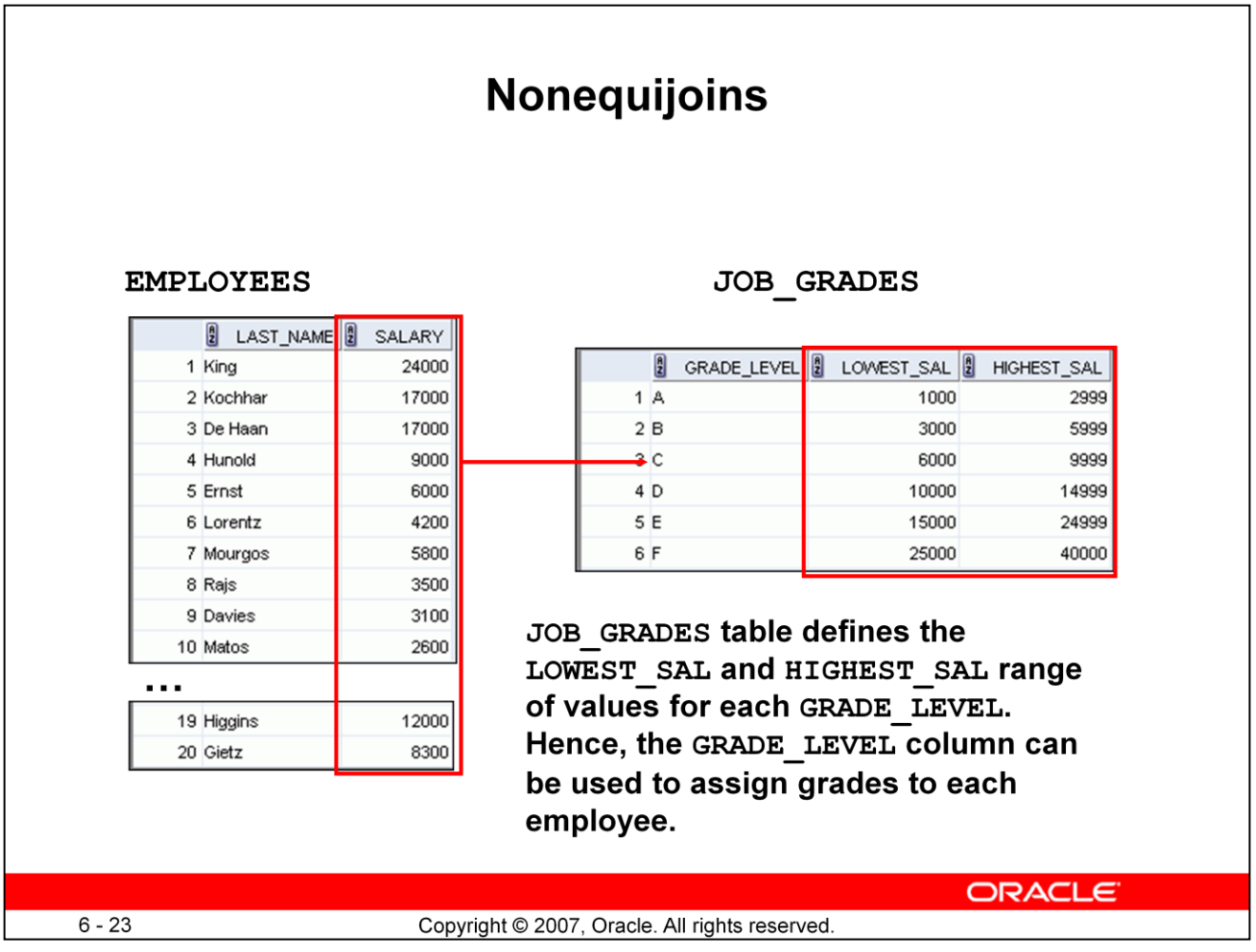
The `ON` clause can also be used to join columns that have different names, within the same table or in a different table.

The example shown is a self-join of the `EMPLOYEES` table, based on the `EMPLOYEE_ID` and `MANAGER_ID` columns.

Note: The parenthesis around the joined columns as in the slide example, `(e.manager_id = m.employee_id)` is **optional**. So, even `ON e.manager_id = m.employee_id` will work.

Lesson Agenda

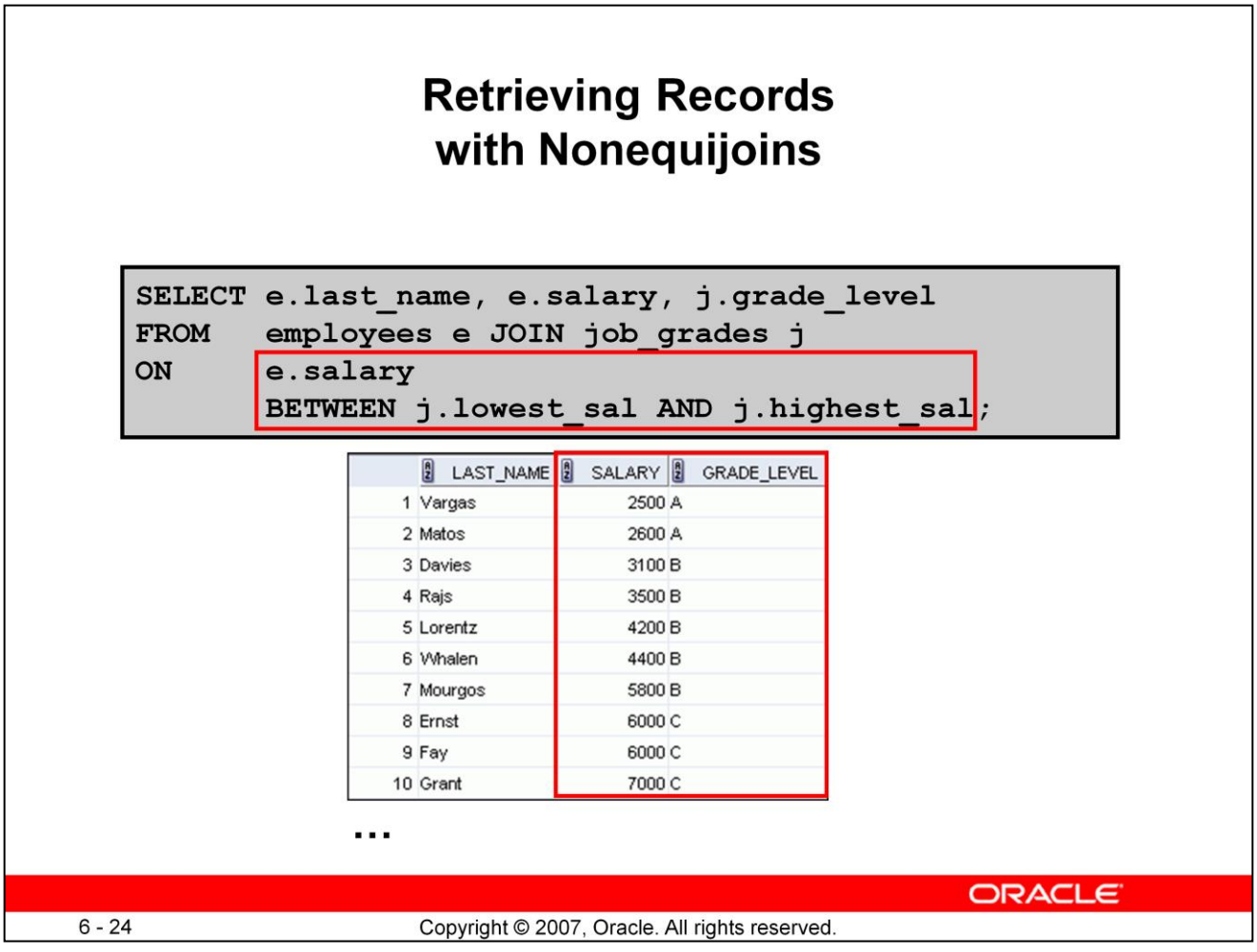
- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join



Nonequijoins

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the `EMPLOYEES` table and the `JOB_GRADES` table is an example of a nonequijoin. The `SALARY` column in the `EMPLOYEES` table ranges between the values in the `LOWEST_SAL` and `HIGHEST_SAL` columns of the `JOB_GRADES` table. Hence, each employee can be graded based on their salary. The relationship is obtained using an operator other than the equality (=) operator.



Retrieving Records with Nonequijoins

The slide example creates a nonequijoin to evaluate an employee's salary grade. The salary must be *between* any pair of the low and high salary ranges.

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

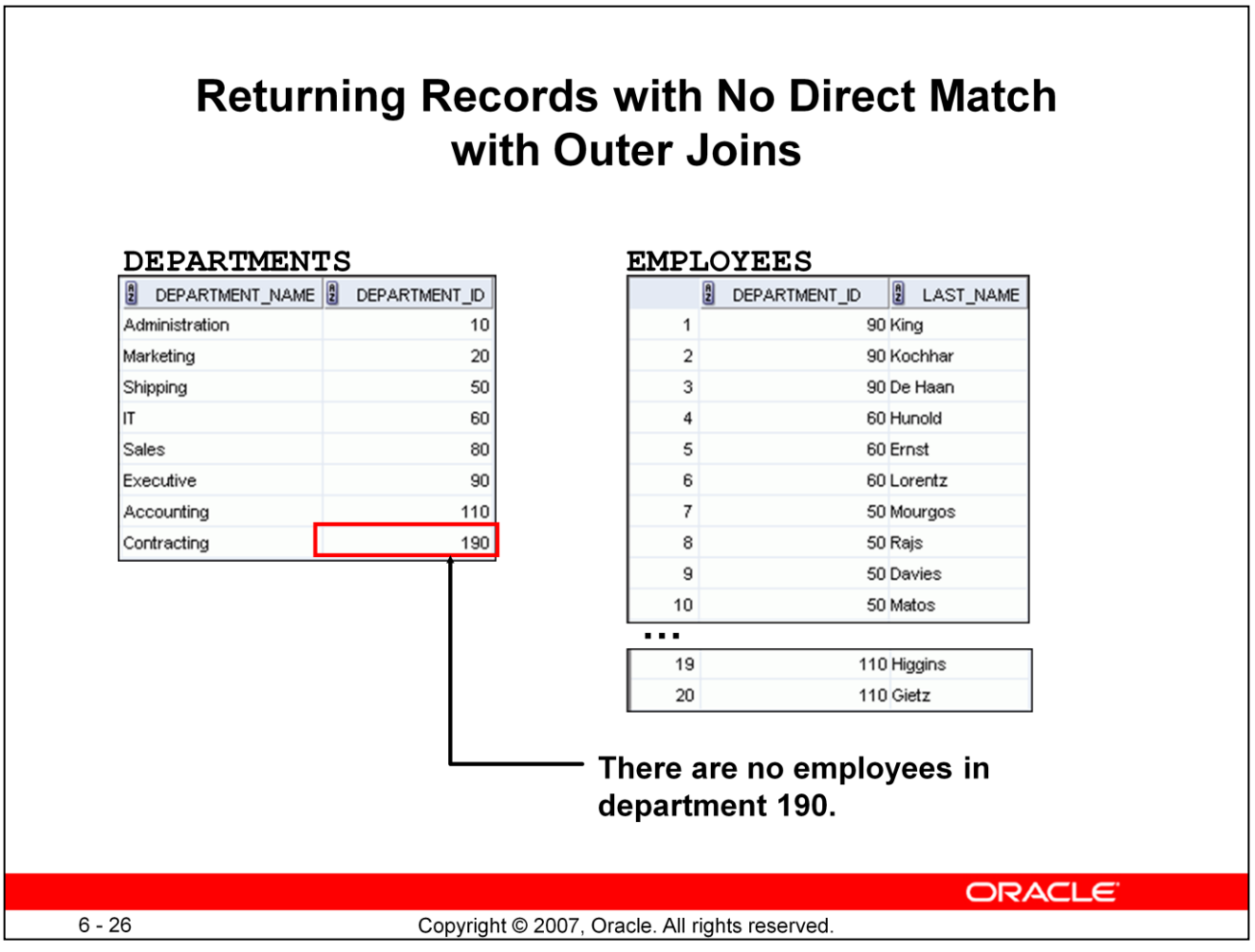
- None of the rows in the `JOB_GRADES` table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

Note: Other conditions (such as <= and >=) can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using the `BETWEEN` condition. The Oracle server translates the `BETWEEN` condition to a pair of `AND` conditions. Therefore, using `BETWEEN` has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join



Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of `EMPLOYEES` and `DEPARTMENTS` tables, department ID 190 does not appear because there are no employees with that department ID recorded in the `EMPLOYEES` table. Therefore, instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) table is called a left (or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

ORACLE

INNER Versus OUTER Joins

Joining tables with the `NATURAL JOIN`, `USING`, or `ON` clauses results in an inner join. Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an outer join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of outer joins:

- `LEFT OUTER`
- `RIGHT OUTER`
- `FULL OUTER`

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping
...			
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

LEFT OUTER JOIN

This query retrieves all rows in the `EMPLOYEES` table, which is the left table, even if there is no match in the `DEPARTMENTS` table.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

ORACLE

RIGHT OUTER JOIN

This query retrieves all rows in the DEPARTMENTS table, which is the right table, even if there is no match in the EMPLOYEES table.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
...			
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

ORACLE

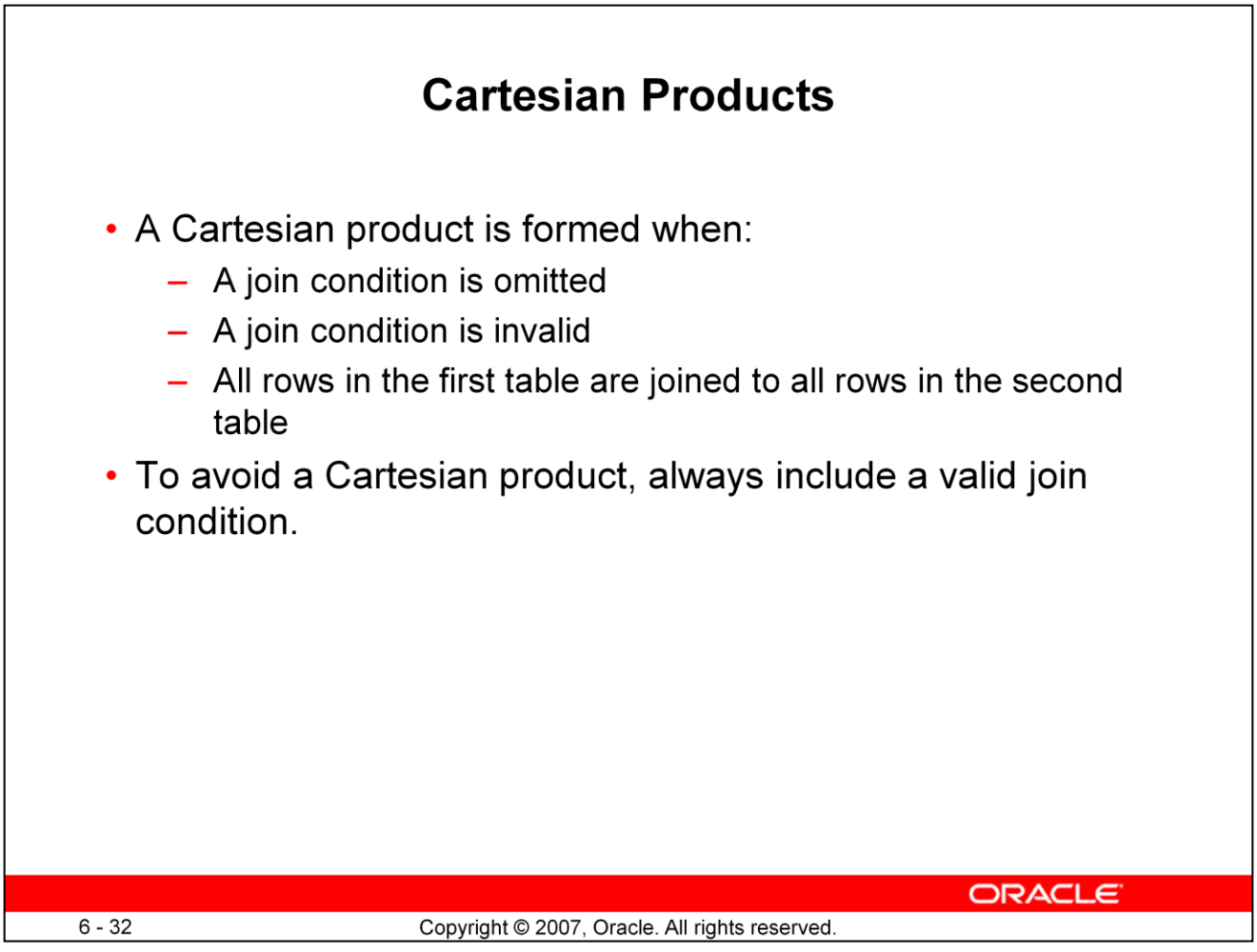
FULL OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequiijoin
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join

ORACLE

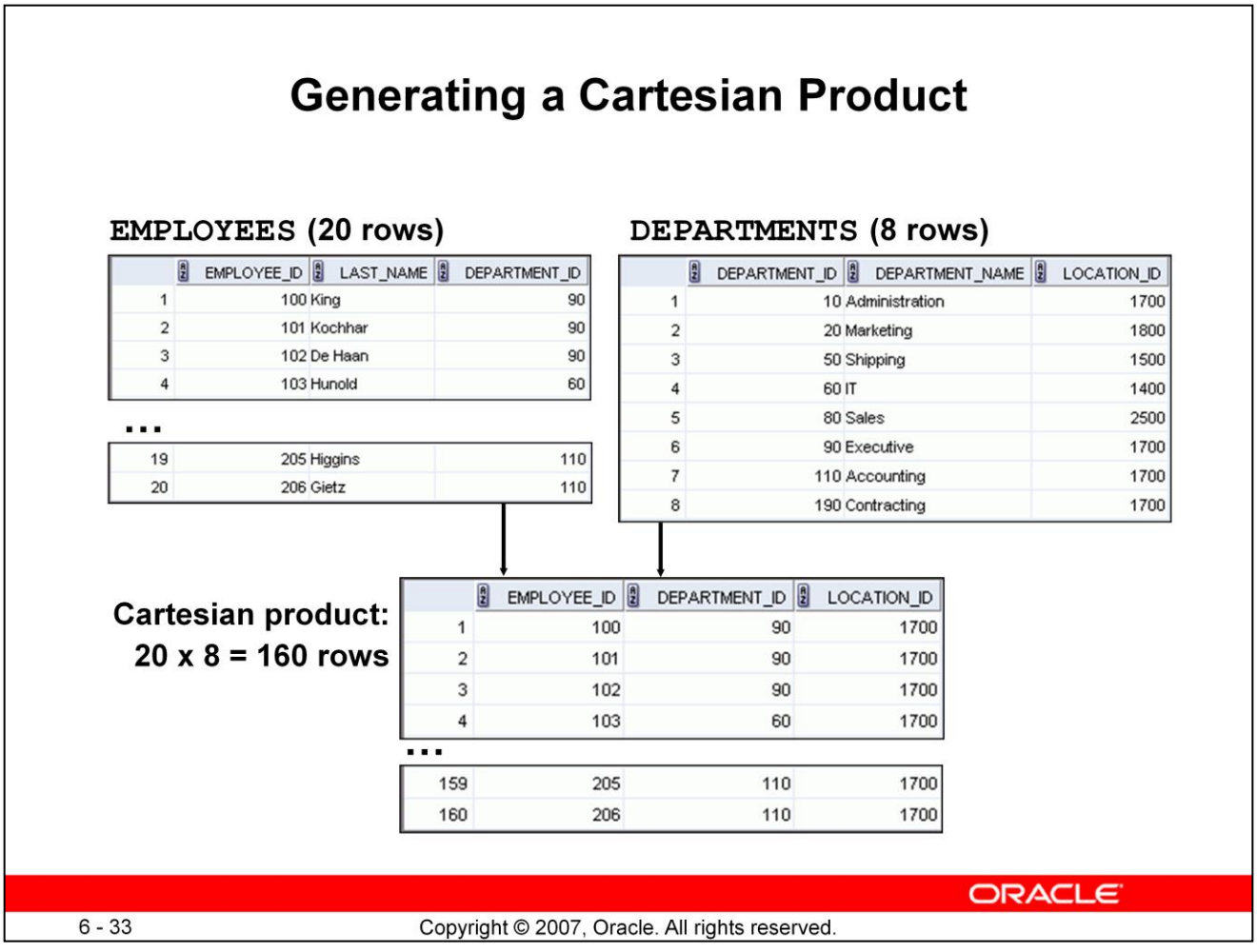


Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product*, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

A Cartesian product tends to generate a large number of rows and the result is rarely useful. You should, therefore, always include a valid join condition unless you have a specific need to combine all rows from all tables.

However, Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.



Generating a Cartesian Product

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee last name and the department name from the `EMPLOYEES` and `DEPARTMENTS` tables. Because no join condition was specified, all rows (20 rows) from the `EMPLOYEES` table are joined with all rows (8 rows) in the `DEPARTMENTS` table, thereby generating 160 rows in the output.

Creating Cross Joins

- The `CROSS JOIN` clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

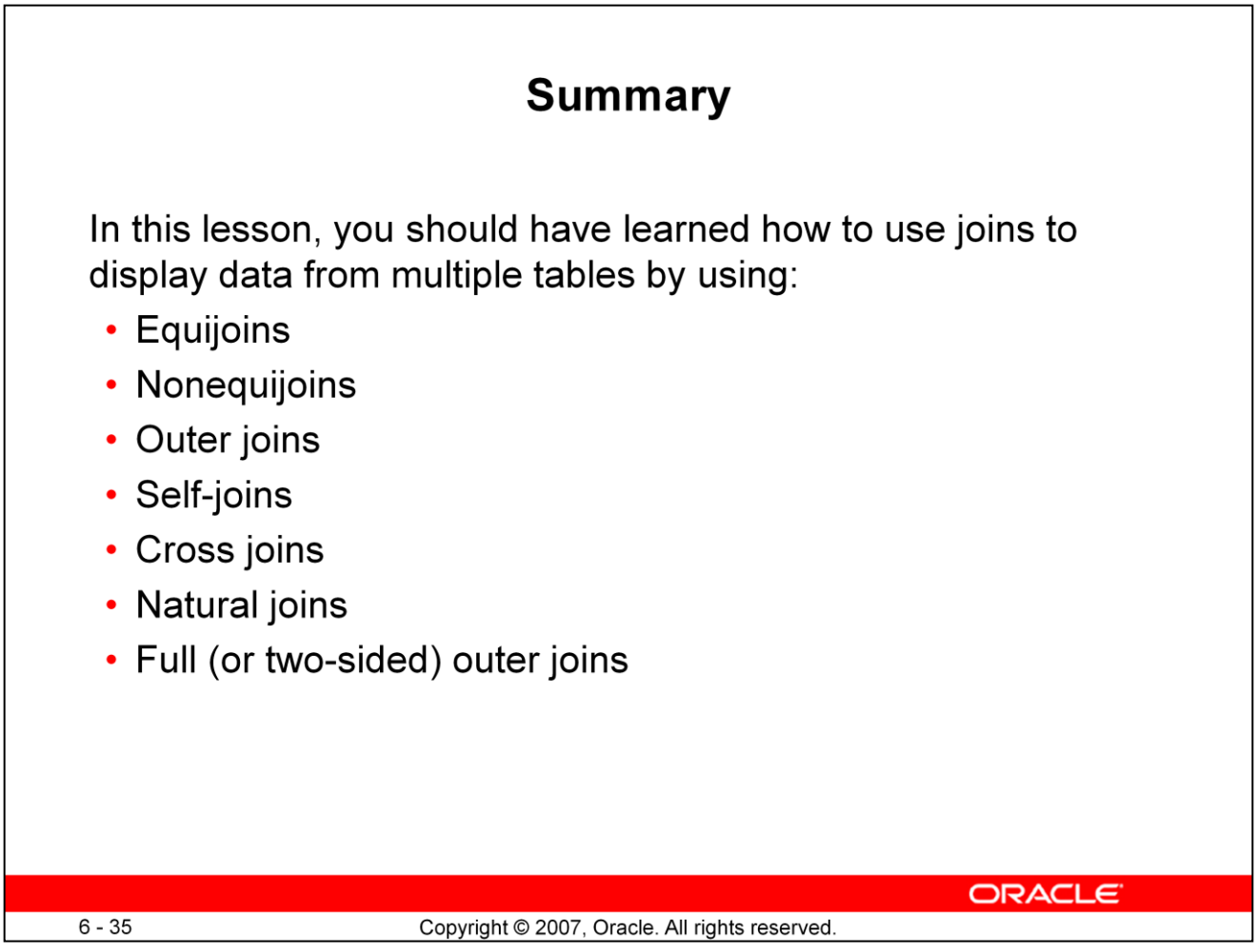
```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
159	Whalen	Contracting
160	Zlotkey	Contracting

ORACLE

Creating Cross Joins

The example in the slide produces a Cartesian product of the `EMPLOYEES` and `DEPARTMENTS` tables.



Summary

There are multiple ways to join tables.

Types of Joins

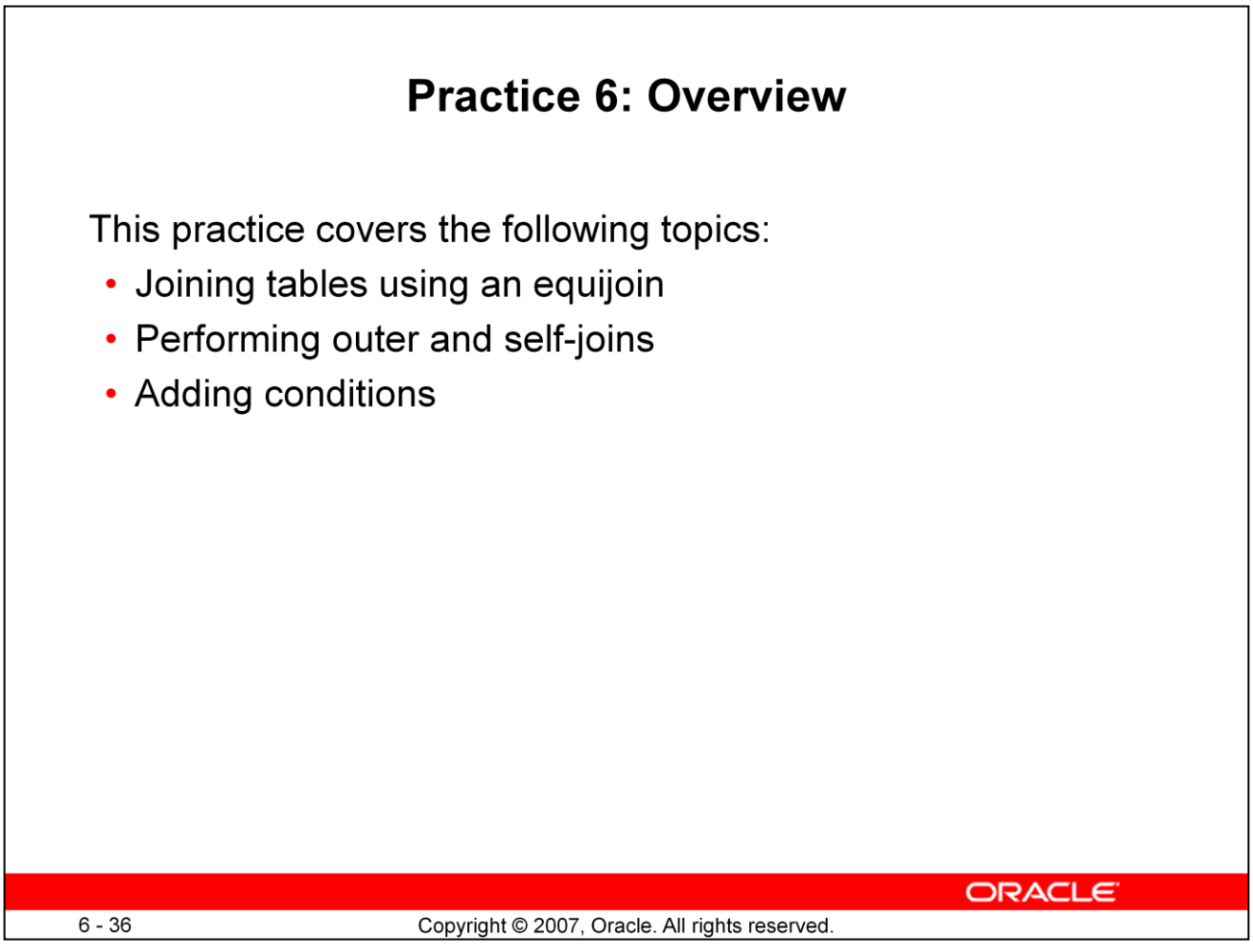
- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

Cartesian Products

A Cartesian product results in the display of all combinations of rows. This is done by either omitting the `WHERE` clause or by specifying the `CROSS JOIN` clause.

Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.
- Table aliases are sometimes mandatory to avoid column ambiguity.



Practice 6: Overview

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999–compliant joins.